Introduction[1] aux bases de données

I] Bases de données

A. Présentation

Notre société utilise de plus en plus de données et un des enjeux de l'efficacité économique mais aussi de la démocratie est d'en maîtriser l'exploitation.
Nous allons décrire dans ce chapitre le modèle de description d'un ensemble de données, les opérations qu'on veut pouvoir effectuer et le langage universel qui permet ces opérations.

1/ Données

Un ensemble de données est avant tout constitué d'information, l'enjeu est d'arriver à trouver une réponse depuis ces ensemble d'information. Quand les données ne sont pas bien structurées cela peut devenir une énigme, comme dans certains jeux.

Pour organiser les données on va utiliser un modèle simple, celui d'un tableau.
Les données concernent plusieurs objets qui ont des propriétés, on se donne à l'avance un ensemble de caractéristiques et on donne, pour chaque objet, la valeur prise pour chaque caractéristique. On obtient un tableau à deux dimensions avec une ligne par objet et une colonne par caractéristique. Les colonnes sont nommées mais les objets sont définis simplement par leur ensemble de valeurs, s'ils ont un nom alors c'est une de leurs propriétés.
On rencontre souvent ce type de structures :

— répertoire (nom, téléphone, adresse, ...)
— fiche de bibliothèque (auteur, titre, année, pagination, ...)
— livre d'état-civil des naissances (père, mère, nom, lieu, ...)
— ...

Voici un exemple simple dont nous nous servirons. Ce sont les vainqueurs des élections présidentielles de la cinquième république.

nom	prénom	études	politique	année	résultat
de Gaulle	Charles	Saint-Cyr	D	1958	78,5
de Gaulle	Charles	Saint-Cyr	D	1964	55,2
Pompidou	Georges	ENS Ulm	D	1969	58,2
Giscard d'Estaing	Valery	X	C	1974	50,8
Miterrand	François	Droit	G	1981	51,8
Miterrand	François	Droit	G	1988	54,0
Chirac	Jacques	ENA	D	1995	52,6
Chirac	Jacques	ENA	D	2002	82,2
Sarkozy	Nicolas	Droit	D	2007	53,0
Hollande	François	ENA	G	2012	51,6
Macron	Emmanuel	ENA	C	2017	66,1

2/ Travailler sur des données

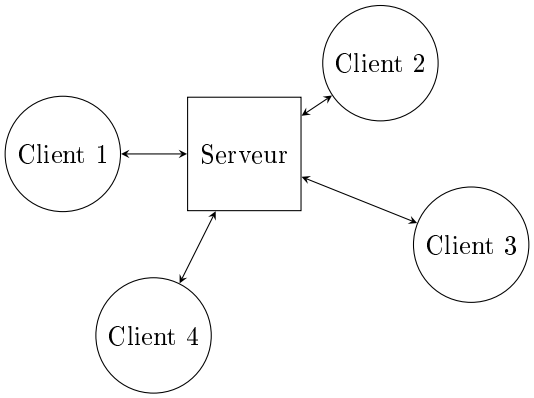
a) Travail en direct

La première possibilité pour traiter ces données, et qui reste certainement la plus courante, est d'utiliser un TABLEUR. C'est une solution très simple qui permet de faire des calculs sur des données quand on n'a pas besoin d'en extraire une partie. On peut trier selon une caractéristique, faire des calculs globaux (statistiques), faire des combinaisons de valeurs mais il est difficile de faire des moyennes partielles, par exemple dans le tableau ci-dessus, calculer la moyenne des résultats par couleur politique.
De plus il est difficile de permettre l'utilisation par plusieurs personnes d'une même base.

b) L'architecture client-serveur

On a choisi d'abstraire la base de données en la plaçant sur un SERVEUR, chaque utilisateur, ou CLIENT, envoie ses demandes, on parle de REQUÊTES, au serveur qui lui répond.

1. Ce poly est celui des collègues du lycée Faidherbe de Lille, merci à eux. Si vous désirez aller voire ce qu'ils font : `http://eric-detrez.fr/`

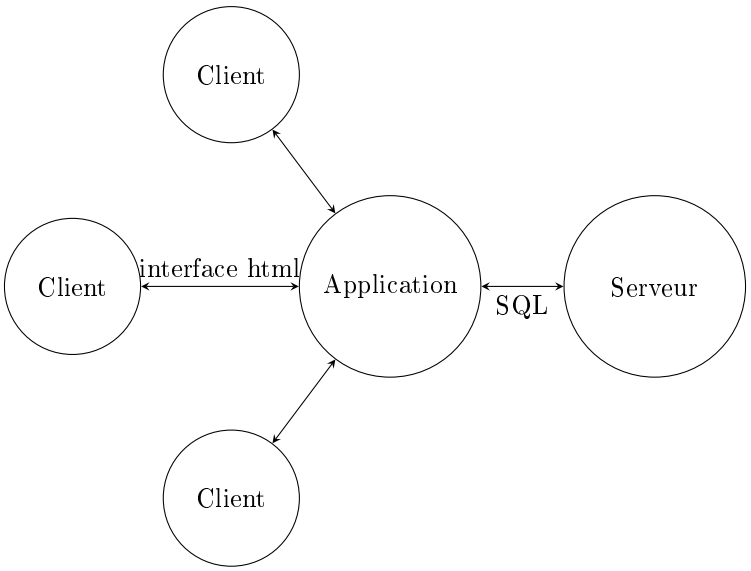


Il ne faut pas se fier à la simplicité apparente de cette structure, les problèmes à gérer sont nombreux et délicats :
- organisation des données,
- rapidité d'accès,
- gestion des autorisations d'accès,
- priorités des accès,
- sauvegarde des modifications,
- gestion des pannes ...

Il faut donc un langage pour établir les requêtes auprès d'un serveur. Ici, un phénomène inhabituel en informatique s'est produit, Cependant, contrairement aux langages de programmation, un standard a été choisi, c'est le langage SQL que nous allons partiellement exposer dans la suite. Ce langage permet de définir, modifier, interroger les différents systèmes du point de vue des utilisateurs. Bien entendu la programmation en interne de ces gros logiciels dépend de l'éditeur (Oracle, SAP, IBM, Microsoft, ...)

c) Pour les utilisateurs

On peut remarquer que SQL n'est pas visible pour les usagers. En effet l'utilisation des bases de données se fait maintenant à travers une architecture TROIS-TIERS. Entre l'usager et la base de données se place l'application qui travaille au dessus du gestionnaire de bases de données.



d) Plan

Dans la suite nous allons d'abord présenter le modèle des tableaux et le vocabulaire associé, puis nous définirons les opérations souhaitées. La partie suivante donnera la traduction opératoire de ces concepts dans le langage SQL.
On introduit enfin les outils statistiques.
La section II] exposera un autre aspect de ce modèle, la possibilité de croiser plusieurs tables de données.

B. Relations

1/ Vocabulaire

La composante de base, le tableau, est appelé **relation**.
On a vu qu'il comportait deux parties :
- les différents titres des colonnes qui sont appelés **attributs** ; on appelle **schéma** (**sort** en anglais) l'ensemble des attributs,

— l'ensemble des lignes est l'**extension** de la relation, une ligne sera nommée **n-uplet** (**tuple** en anglais).

On nommera $A_1, A_2, \ldots, A_p$ les attributs.
L'ensemble des valeurs possibles d'un attribut A est son domaine $\mathrm{Dom}(A)$.
Chaque n-uplet est donc un élément de $\mathrm{Dom}(A_1) \times \mathrm{Dom}(A_2) \times \cdots \times \mathrm{Dom}(A_p)$.

2/ Contraintes

La définition énoncée ci-dessus implique des contraintes. Certaines de ces conséquences devront être assurées par l'utilisateur d'autres seront contrôlées par le logiciel.

a) Les attributs forment un ensemble

— Il n'y a pas d'attribut en double : il ne peut être question qu'il y ait deux attributs de même nom, ils risqueraient d'être affectés de valeurs différentes dans des n-uplets.
— L'ordre des attributs n'est pas fixé :
on ne parle pas du premier attribut mais de l'attribut `nom_attribut` (par exemple).
— En conséquences les différentes données ne sont donc pas des n-uplets au sens informatique du terme. Ce sont plutôt des fonctions depuis l'ensemble des attributs vers l'union des domaines avec la condition que $t[A_i]$ appartient au domaine $\mathrm{Dom}(A_i)$ (remarquer que l'on écrit avec des crochets plutôt que des parenthèses).
— On peut alors restreindre une ligne à un sous-ensemble d'attributs : si S est une partie de $\{A_1, A_2, \ldots, A_p\}$ on notera $t[S]$ l'ensemble des valeurs prises par le n-uplet t pour les attributs appartenant à S, par exemple $t[A_4, A_5, A_2]$.

b) Les lignes forment un ensemble

— Il n'y a pas de ligne en double : si $t[A_1, \ldots, A_p] = t'[A_1, \ldots, A_p]$ alors $t = t'$; cela correspond à l'aspect fonction des n-uplets.
— L'ordre des lignes n'est pas fixé.

Bien entendu une présentation des données dans un tableau aura un ordre des attributs et un ordre des n-uplets mais ces ordres de présentations sont le résultats des choix de l'utilisateur ou de l'ordre physique des données.

3/ Clés

La contrainte de non-répétition des n-uplets est très importante ; la répétition de données induirait des problème de rapidité d'accès et fausserait les calculs. Pour cela les bases de données réelles contiennent un concept de **clé** qui doit être pensé dès la conception des bases de données. On verra qu'il a aussi une grande importance lors de l'utilisation de relations multiples.

Définition : Super-clé
Une super-clé d'une relation r est un ensemble S d'attributs tel que

$$\forall t, t' \in R \quad \big(t[S] = t'[S]\big) \Rightarrow (t = t')$$

Les contraintes imposent que l'ensemble de tous les attributs est toujours une super-clé.
Dans l'exemple ci-dessus {`année`}, {`nom`, `résultat`}, {`prénom`, `année`} sont des super-clés.

Définition : Clé candidate
Une clé candidate d'une relation r est une super-clé minimale pour l'inclusion.

K est donc une clé candidate si

1. K est une super-clé
2. Pour tout K' inclus dans K, K' n'est pas une super-clé.

Une super-clé qui ne contient qu'un seul attribut est toujours une clé candidate.
— {`nom`, `résultat`} n'est pas une clé candidate.
— {`année`} est une clé candidate.
— Lorsque vous vous identifiez sur un site sur lequel vous êtes inscrit, l'identifiant + le mot de passe est une clé candidate.

Définition : Clé primaire
Parmi les clés candidates on en choisit une : c'est la clé primaire.

Pour indiquer la clé primaire on souligne les attributs correspondants dans la table.
Indiquer au système une clé primaire pour chaque table permet une indexation des données à l'aide de cette clé, ce qui renforce l'efficacité des procédures d'interrogation de la table.
Nous connaissons déjà ce genre d'attributs : numéro de sécurité sociale, plaque minéralogique, numéro de série ...
Pour obtenir une clé primaire efficace il est recommandé introduire un attribut, souvent nommé `id`, dans la définition de la relation.
Ce sera un entier qu'on incrémentera à chaque ajout d'un n-uplet.

id	nom	prénom	études	politique	année	résultat
1	de Gaulle	Charles	Saint-Cyr	D	1958	78,5
2	de Gaulle	Charles	Saint-Cyr	D	1964	55,2
3	Pompidou	Georges	ENS Ulm	D	1969	58,2
4	Giscard d'Estaing	Valery	X	C	1974	50,8
5	Miterrand	François	Droit	G	1981	51,8
6	Miterrand	François	Droit	G	1988	54,0
7	Chirac	Jacques	ENA	D	1995	52,6
8	Chirac	Jacques	ENA	D	2002	82,2
9	Sarkozy	Nicolas	Droit	D	2007	53,0
10	Hollande	François	ENA	G	2012	51,6
11	Macron	Emmanuel	ENA	C	2017	66,1

C. Algèbre relationnelle simple

Pour modéliser les requêtes on va considérer qu'une relation n'est qu'une relation parmi toutes les relations possibles et on a va modéliser la construction de relations à partir d'autres relations.
Depuis 1970 deux modèles abstraits de structure des données et d'extraction ont été étudiés : l'algèbre relationnelle et le calcul relationnel.
L'**algèbre relationnelle** formalise les opérations qui doivent être faites, le **calcul relationnel** décrit ce que l'on veut obtenir.
Ces deux modèles sont équivalents et ont servi de base théorique aux logiciels de base de données que les progrès de l'informatique ont permis d'écrire de manière efficace.
Nous allons étudier l'un de ces modèles : l'algèbre relationnelle.

1/ Définition

L'algèbre relationnelle est un ensemble d'opérateurs que l'on peut appliquer à des relations, et dont le résultat est une relation.
Comme le résultat est toujours une relation on pourra combiner ces opérateurs : on forme ainsi des **requêtes** élaborées.
L'objectif est de pouvoir exprimer toute requête raisonnable par une combinaison d'opérateurs élémentaires.
Exemples de requêtes sur le tableau donné en exemple.

- Quels sont les présidents de droite ?
- Quels sont les présidents qui sortent de l'ENA ?
- Quels présidents ont gouverné plus de 10 ans ?

2/ Opérateurs ensemblistes

Un premier type d'opérateur combine 2 relations qui ont le même schéma ; on les utilisera surtout pour assembler les résultats d'autres requêtes.
r et r' sont deux relations ayant le même schéma (c'est-à-dire les mêmes attributs).

1. $r \cup r'$ est la relation de même schéma dont les n-uplets appartiennent à r **ou** à r'.
2. $r \cap r'$ est la relation de même schéma dont les n-uplets appartiennent à r **et** à r'.
3. $r \setminus r'$ est la relation de même schéma dont les n-uplets appartiennent à r **mais pas** à r'.

3/ Opérateurs spécifiques

Les opérateurs étudiés ici sont ceux qui utilisent la structure des relations. Dans cette partie nous n'étudions que les opérateurs unaires, qui s'appliquent à une relation unique.
Par exemple la question "Quels sont les présidents qui sortent de l'ENA ?" se traduit par :
donner les noms (projection) des présidents dont les études se sont passées à l'ENA (sélection).

Le **RENOMMAGE** consiste à renommer un attribut.
Ce sera utile lors de produits de tables lorsque deux tables ont des attributs portant le même nom mais correspondent à des données différentes.
Si $A \in R$ où R est le schéma de r et $B \notin R$ on note $\rho_{A\to B}(r)$ la relation obtenue en changeant le nom de l'attribut A en B.

1. Si R est le schéma de r avec $A \in R$, le schéma de $\rho_{A\to B}(r)$ est $(R \setminus \{A\}) \cup \{B\}$.
2. L'extension de $\rho_{A\to B}(r)$ est $\{t \ ; \ \exists s \in r, t[B] = s[A], \forall C \in R \setminus \{A\}, \ t[C] = s[C]\}$.

La **SELECTION** (ou restriction) consiste à ne garder que les n-uplets qui vérifie une propriété.
On note $\sigma_F(r)$ la relation extraite de r par la propriété F.

1. $\sigma_F(r)$ a le même schéma que r

2. L'extension de $\sigma_F(r)$ est $\{t \in r \ ; \ t \text{ vérifie } F\}$

On notera, pas abus d'écriture, $\sigma_F(r) = \{t \in r \ ; \ t \text{ vérifie } F\}$ sans noter explicitement le schéma.

Les propriétés élémentaires sont de la forme $P\,\theta\,Q$ où θ un opérateur de comparaison ($=$, $<$, $\leq$, $>$, $\geq$) et P et Q des expression construites à partir des attributs et des constantes avec des fonctions usuelles.

Une propriété composée avec des connecteurs logiques correspond à des unions et intersections et peut être écrite directement.
Par exemple $\sigma_{F\,et\,F'}(r) = \sigma_F(r) \cap \sigma_{F'}(r)$.

Exemple : F est la condition "prénom = François", $\sigma_F(r)$ est la relation

id	nom	prénom	études	politique	année	résultat
5	Miterrand	François	Droit	G	1981	51,80
6	Miterrand	François	Droit	G	1988	54,0
10	Hollande	François	ENA	G	2012	51,6

La **PROJECTION** consiste à ne garder qu'une partie des attributs. On note $\pi_X(r)$ la relation extraite de r avec les restriction des n-uplets à l'ensemble X.

1. $\pi_X(r)$ admet X pour schéma.
2. L'extension de $\pi_X(r)$ est $\{t[X];\ t \in r\}$

On notera, pas abus d'écriture, $\pi_X(r) = \{t[X];\ t \in r\}$ sans noter explicitement le schéma et en assimilant la restriction à X avec l'image $t[X]$.
Exemple : si X est l'ensemble {prénom, nom, résultat}, $\pi_X(r)$ est la relation

nom	prénom	résultat
de Gaulle	Charles	78,5
de Gaulle	Charles	55,2
Pompidou	Georges	58,2
Giscard d'Estaing	Valery	50,8
Miterrand	François	51,8
Miterrand	François	54,0
Chirac	Jacques	52,6
Chirac	Jacques	82,2
Sarkozy	Nicolas	53,0
Hollande	François	51,6
Macron	Emmanuel	66,1

Comme la projection doit être une relation les lignes ne sont pas répétées :
si X={étude}, $\pi_X(r)$ donne

études
Saint-Cyr
ENS Ulm
Polytechnique
Fac de droit
ENA

D. SQL

1/ Introduction

Nous venons de décrire les premiers opérateurs de l'algèbre relationnelle.
On peut les comparer aux instructions de bases en programmation.
Pour les langages de requêtes il s'est produit un événement inhabituel : une norme universelle a été établie qui permet d'écrire des requêtes de la même façon quel que soit le logiciel : c'est **SQL**, pour Structured Query Langage.
Le langage SQL reprend la structure de l'algèbre relationnelle en y ajoutant des moyens de calculs et autres améliorations. Il le fait dans une formulation plus proche d'un langage humain (l'anglais) en structurant les requêtes dans un modèle simple.
Bien entendu chaque éditeur optimise les opérateurs pour donner des réponses le plus rapidement possible, ajoute des améliorations supplémentaires mais presque tous contiennent le langage tel qu'il est normalisé.

2/ Instructions de base

Les représentations des relations se font avec le modèle du tableau.
En SQL on parlera de
- **tables** à la places de relations
- **colonnes** à la places d'attributs
- **lignes** à la places de n-uplets.

La forme de base d'une requête en SQL est une composition d'une sélection et d'une projection.

```
FROM table1
WHERE condition;
```

Les mots-clés de SQL sont usuellement écrits en majuscule mais ce n'est pas obligatoire

SELECT correspond à la projection, les attributs ou colonnes à afficher sont énumérés et séparés par une virgule. Il y a toujours une projection.
Si on ne veut pas effectuer de projection, on doit indiquer que l'on veut toutes les colonnes. On peut le faire par le symbole * qui signifie "tout".

FROM sélectionne le nom de la table à utiliser.
Cela deviendra plus signifiant lorsqu'on utilise plusieurs tables.

WHERE correspond à la sélection (attention : `SELECT` ne désigne pas la sélection).
La condition peut être composée à l'aide des connecteurs logiques `AND`, `OR` ou `NOT`.
Les comparateurs sont `=`, `>`, `<`, `>=`, `<=`, `!=`.
Les chaînes de caractères sont entourées de guillemets simples ou doubles.
La clause `WHERE` est facultative. On peut ne pas faire de sélection.

AS On peut renommer une colonne avec AS suivi du nouveau nom.

UNION, INTERSECT, EXCEPT Si on veut combiner plusieurs requêtes on peut les assembler avec `UNION` (pour $\cup$), `INTERSECT` (pour $\cap$) ou `EXCEPT` (pour $\setminus$).

Exemple : quels sont les noms et prénoms des présidents ayant été élus avant 1970 ou étant passés par l'ENA ? La question en algèbre relationnelle s'écrit

$$\pi_{\text{nom,prénom}}\big(\sigma_{\text{année} < 1970}(r)\big) \cup \pi_{\text{nom,prénom}}\big(\sigma_{\text{études} = \text{ENA}}(r)\big)$$

En SQL cela se traduit directement par

```
  SELECT nom, prénom
  FROM élections
  WHERE année < 1970
UNION
  SELECT nom, prénom
  FROM élections
  WHERE études = 'ENA';
```

ou, plus simplement,

```
SELECT nom, prénom
FROM élections
WHERE année < 1970 OR études = 'ENA';
```

Dans tous les cas on obtient

nom	prénom
de Gaulle	Charles
de Gaulle	Charles
Pompidou	Georges
Chirac	Jacques
Chirac	Jacques
Hollande	François
Macron	Emmanuel

SQL ne respecte pas la contrainte d'unicité des lignes.
Si on veut éviter les redondances il faut ajouter `DISTINCT` à `SELECT` :

```
SELECT DISTINCT nom, prénom
FROM élections
WHERE année < 1970
OR études='ENA'
```

donne

nom	prénom
de Gaulle	Charles
Pompidou	Georges
Chirac	Jacques
Hollande	François

3/ Fonctions de présentation

SQL permet d'améliorer la présentation des résultats en triant selon un attribut avec ORDER BY. Le résultat est trié selon l'ordre croissant.
Si l'on veut un ordre décroissant on le spécifie par `ORDER BY colonne DESC`
Cette instruction est placée à la fin de la requête.

On peut aussi limiter le nombre de résultats en faisant suivre la requête par LIMIT n.

a) Exemple

Si on veut le nom, l'année et le score des 2 meilleurs résultats des élections

```
SELECT nom, année, résultat
FROM élections
ORDER BY résultat DESC
LIMIT 2
```

donne

nom	année	résultat
Chirac	2002	82,2
de Gaulle	1958	78,5

E. Fonctions sur les attributs

1/ Fonctions mathématiques

Il sera souvent utile d'effectuer des calculs à l'aide des valeurs renvoyées par une requête.
On peut utiliser les 4 fonctions arithmétiques de base : `+, -, *, /`
il est recommandé de renommer le résultat.
Par exemple, si une table contient les attributs `population` et `surface` on peut calculer la densité
`SELECT nom, population/surface AS densité, ...`
Remarque : si les colonnes utilisées ont des valeurs entières, les opérations sont à valeurs entières. En particulier la division est alors la division euclidienne : `7/2` renverra `3`. Pour obtenir la division usuelle il suffit de convertir au moins un des termes en flottant :
`(population + 0.0)/surface AS densité`.

Les gestionnaires de bases de données peuvent contenir d'autres fonctions, **sin**, **exp**, ...(mais ce n'est pas le cas de SQLite).

2/ Fonctions statistiques

Les fonction ci-dessus permettent de calculer un résultat pour chaque ligne filtrée par la sélection : on crée en fait une nouvelle colonne.
On peut aussi utiliser le résultat d'une requête à des fins statistiques en calculant un résultat à partir de l'ensemble des données filtrées. Le résultat est alors table à une seule valeur calculée (ou plusieurs si on demande plusieurs calculs statistiques). Il n'est pas pertinent de demander l'affichage des attributs : quelle valeur serait donnée ?
Les fonctions disponibles sont

1. le décompte, c'est-à-dire le nombre de lignes : COUNT
2. le maximum des éléments dans une colonne : MAX
3. le minimum des éléments dans une colonne : MIN
4. la somme des éléments d'une colonne : SUM
5. la moyenne des éléments d'une colonne (sum/count) : AVG

Comme la table renvoyée par une fonction statistique n'a qu'une ligne et une colonne, sa valeur peut être utilisée dans les expressions de sélection.

```
SELECT AVG(résultat)
FROM élections;
```

donne 59,09.

```
SELECT nom, année,résultat
FROM élections
WHERE résultat  > (SELECT AVG(résultat)
                   FROM élections);
```

renvoie les résultats

nom	année	résultat
de Gaulle	1958	78,5
Chirac	2002	82,2
Macron	2017	66,1

3/ Agrégations

On peut affiner les calculs statistiques en les calculant pour les différentes parties d'un découpage de la table.

a) Partition

Si on se donne un attribut B (ou un ensemble d'attributs) on peut partitionner la table selon les valeurs prises par B.

La table suivante correspond à la partition selon **études**.
On remarque que l'ordre peut être changé.

id	nom	prénom	études	politique	année	résultat
1	de Gaulle	Charles	Saint-Cyr	D	1958	78,5
2	de Gaulle	Charles	Saint-Cyr	D	1964	55,2
3	Pompidou	Georges	ENS Ulm	D	1969	58,2
4	Giscard d'E.	Valery	X	C	1974	50,8
5	Miterrand	François	Droit	G	1981	51,8
6	Miterrand	François	Droit	G	1988	54,0
9	Sarkozy	Nicolas	Droit	D	2007	53,0
11	Macron	Emmanuel	ENA	C	2017	66,1
10	Hollande	François	ENA	G	2012	51,6
8	Chirac	Jacques	ENA	D	2002	82,2
7	Chirac	Jacques	ENA	D	1995	52,6

b) Statistiques

On peut alors appliquer une fonction statistique à chacune des parties.
On note ${}_B\gamma_{f(A)}(r)$ cette opération si f est la fonction statistique appliquée à l'attribut A..
La table obtenue admet 2 attributs :

1. un attribut B qui prend les valeurs de B dans la table
2. un attribut calculé : pour chaque valeur b prise par B on calcule $f(A)$ pour la table r où l'on a sélectionné $B = b$.

${}_{\text{études}}\gamma_{\text{max(année)}}(r)$ donne la dernière année d'élection selon les écoles

études	année
ENA	2017
Saint-Cyr	1964
Droit	2007
ENS Ulm	1969
X	1974

${}_{\text{politique}}\gamma_{\text{moyenne(résultat)}}(r)$ donne la moyenne des résultats selon l'orientation politique

politique	résultat
C	58,45
D	63,28
G	52,47

— On peut faire plusieurs calculs statistiques par partie, ce qui donnera plus d'une colonne de calculs.
— On peut partitionner selon plusieurs attributs : chaque partie est alors définie par une même valeur pour chacun des attributs choisis.
— Un calcul statistique global, vu auparavant, correspond à une partition selon un ensemble vide d'attributs : il n'y a alors qu'une seule partie.

4/ Agrégations en SQL

Dans SQL on regroupe les données par `GROUP BY colonne1, colonne2, ..., colonnep`.
Cette instruction est placée après l'éventuelle clause `WHERE`.
L'instruction `SELECT` ne doit contenir que les colonnes qui servent au regroupement et les fonctions statistiques. On rappelle qu'il est utile de nommer ces dernières.

La requête ci dessus s'écrit

```
SELECT politique, AVG(résultat) AS résultatMoyen
FROM élections
GROUP BY politique;
```

Autre exemple : calcul des moyennes des résultats des mandats par président.

```
SELECT nom, AVG(résultat) AS moyenne
FROM élections
GROUP BY nom;
```

nom	moyenne
de Gaulle	66,9
Pompidou	58,2
Giscard d'Estaing	50,8
Miterrand	52,9
Chirac	67,4
Sarkozy	53,0
Hollande	51,6
Macron	66,1

On peut toujours filtrer avant les regroupements (en amont des calculs) :

```
SELECT nom, AVG(résultat) AS moyenne
FROM élections
WHERE politique ='D'
GROUP BY nom;
```

nom	moyenne
de Gaulle	66,9
Pompidou	58,2
Chirac	67,4
Sarkozy	53,0

On peut aussi filtrer en utilisant les attributs calculés. Il faut donc imbriquer les requêtes.

```
SELECT nom,  moyenne
FROM (SELECT nom, AVG(résultat) AS moy
      FROM élections
      WHERE politique ='D'
      GROUP BY nom)
WHERE  moyenne < 60;
```

nom	moy
Pompidou	58,2
Sarkozy	53,0

SQL permet de simplifier cette construction en autorisant une sélection après calcul à l'aide de l'instruction `HAVING` qui se place après le `GROUP BY`.

```
SELECT nom, AVG(résultat) AS moy
FROM élections
WHERE politique ='D'
GROUP BY nom
HAVING moyenne < 60;
```

II] Bases de données composées

A. Tables multiples

1/ Utilité

Dans le chapitre précédent on a décrit ce qui peut se faire avec un formulaire unique qui doit contenir tout ce qui est pertinent.
La plupart du temps une base de données sera composées de plusieurs tables. Pour illustrer l'utilité de cette structure et la construction d'une base, nous allons utiliser l'exemple d'une base de données concernant les notes de colles d'un lycée. Le cadre sera celui d'une décomposition selon le modèle **entités-associations**, parfois appelé entités-relations.

Si on veut pouvoir gérer les notes de colles on voit tout de suite un ensemble d'attributs nécessaires :
colleur, matière, étudiant, note, date, heure.
Cependant, pour pouvoir identifier les étudiants, il faudra certainement plusieurs attributs : le nom, le prénom, la classe et d'autres caractéristiques. Ces informations sont alors inutilement répétées pour chaque colle passée par l'étudiant ; il en est de même pour les colleurs. Pour éviter cette redondance on organise les informations dans des tables séparées.

2/ Entités

On commence donc par définir deux tables, c pour les colleurs et e pour les étudiants.
Ces tables décrivent des **entités**, c'est-à-dire des objets identifiables qui ont des caractéristiques communes, ici les colleurs ou les étudiants.
Un des gros intérêts de cette séparation est que l'on peut modifier les caractéristiques d'un étudiant (ou d'un colleur) en ne faisant qu'une seule modification de table, les caractéristiques ne sont pas répétées.
Le plus souvent chaque n-uplet sera repéré par un identifiant qui sert de clé primaire, dans nos tables c'est la valeur de l'attribut **id**.

TABLE 1 – Table des colleurs : c

id	**nom**	**prénom**	**établissement**	...
1	Martin	Edgar	Jean Bart	...
2	Durand	Valentin	César Baggio	...
3	Van Oyden	Lucille	Faidherbe	...
...	...	...	...	...

TABLE 2 – Table des étudiants : e

id	**nom**	**prénom**	**classe**	...
1	Aernout	Ludovic	PCSI1	...
2	Camblor	Patricia	MPSI3	...
3	Courbois	Éloïse	MP1	...
...	...	...	...	...

3/ Associations

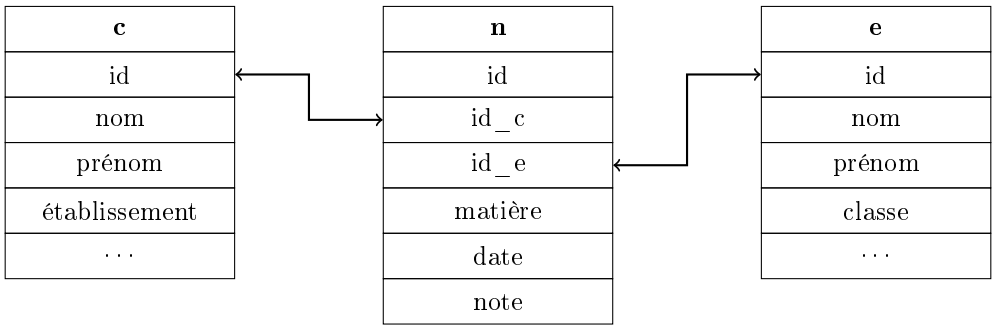
Une note de colle est le résultat d'une rencontre d'un colleur et d'un étudiant on va définir la tables des notes qui va contenir les références aux colleur et étudiant ainsi que les caractéristiques de la colle. C'est une table d'**association**, elle établit une liaison entre entités.
Dans cette table, deux attributs, **id_c** et **id_e** doivent prendre pour valeur un indice existant respectivement dans la table c et dans la table e comme valeurs de l'attribut **id** qui est la clé primaire des tables. Une telle dépendance définit une clé **étrangère**. Une valeur d'une clé étrangère **doit** exister dans la table à laquelle elle réfère.

TABLE 3 – Notes des colles : la table n

id	**id_c**	**id_e**	**matière**	**date**	**heure**	**note**
1	2	1	Math	24/11/2019	17h00	13
2	2	3	Math	24/11/2019	13h00	17
3	3	1	Phys	24/11/2019	17h00	14
...	...	...	...	...	...	

4/ Représentation des schémas de la base

L'égalité, structurelle, entre une clé étrangère et l'attribut auquel elle réfère sera souvent indiquée lors de la représentation des schémas des tables par une liaison, doublement fléchée, entre les attributs qui doivent être égaux.



Structure de la base de données

B. Utilisation de plusieurs tables

1/ Produit

Le premier moyen d'assembler deux relations est d'en faire le produit cartésien.
On crée ainsi une nouvelle table en général beaucoup plus importante.

Définition : Produit de deux relations
Si r est une relation de schéma R et r' est une relation de schéma R' avec R et R' disjoints

alors le produit de r et r' est la relation $r \times r'$ de schéma $R \cup R'$
et dont l'extension est l'ensemble des fonctions définies sur $R \cup R'$ dont les restrictions à R et à R' appartiennent respectivement à l'extension de r et à l'extension de r'.
L'abus usuel d'écriture donne $r \times r' = \{u \ ; \ u[R] \in r, \ u[R'] \in r'\}$

La condition $R \cap R' = \emptyset$ n'est pas très contraignante, il suffit de renommer les noms des attributs ; en général on notera **r.xxx** tous les attributs de r et **r'.xxx** tous les attributs de r' pour éviter les homonymies.
En SQL, le produit est simplement engendré en listant les tables à multiplier dans la clause FROM séparées par des virgules.
Par exemple le produit de c et de e revient à considérer toutes les rencontres possibles entre un colleur et un étudiant.

TABLE 4 – Produit des tables : $c \times e$

c.id	c.nom	c.prénom	c.établissement	...	e.id	e.nom	e.prénom	e.classe	...
1	Martin	Edgar	Jean Bart	...	1	Aernout	Ludovic	PCSI1	...
1	Martin	Edgar	Jean Bart	...	2	Camblor	Patricia	MPSI3	...
1	Martin	Edgar	Jean Bart	...	3	Courbois	Éloïse	MP1	...
...	...	...	...	...	...	...	...	...	...
2	Durand	Valentin	César Baggio	...	1	Aernout	Ludovic	PCSI1	...
2	Durand	Valentin	César Baggio	...	2	Camblor	Patricia	MPSI3	...
...	...	...	...	...	...	...	...	...	...

2/ Produit amélioré

Le produit ci-dessus ne représente rien sauf des paires colleur-étudiant.
Si on veut les notes de colles pour tous les étudiants, on peut faire le produit des tables n et e en imposant la condition d'égalité. Par exemple la table de toutes les notes de tous les étudiants est obtenue par la requête SQL suivante.

```
SELECT e.nom, e.classe, n.matiere, n.date, n.note
FROM e, n
WHERE  e.id = n.id_e
```

Si on veut obtenir de plus le nom du colleur on doit utiliser les trois tables.

```
SELECT e.nom, e.classe, c.nom, n.matiere, n.date, n.note
FROM c, e, n
WHERE  c.id = n.id_c AND e.id = n.id_e
```

Bien entendu on peut ajouter des sélections. Les notes, avec le colleur, des étudiants de PCSI3 en chimie sont obtenues par

```
SELECT e.nom, e.classe, c.nom, n.matiere, n.date, n.note
FROM c, e, n
WHERE  c.id = n.id_c AND e.id = n.id_e
                     AND e.classe = "PCSI3"
                     AND n.matière = "Chim"
```

3/ Jointure

La réponse donnée ci-dessus n'est pas totalement satisfaisante : elle donne certes le bon résultat mais on écrit la requête en écrivant une sélection qui mélange une condition qui n'est pas facultative, celle provenant de la structure de la base, avec des critères de sélection qui dépendent de la question posée (clase, matière, ...).
La construction de la table produit ne contenant que les n-uplets respectant les conditions structurelles est donc fondamentale, c'est la **jointure** (en fait equi-jointure).

Définition : Jointure
Si r et r' sont des relations et si F est une condition sur $r \times r'$ alors la jointure de r et r' selon F est la relation $r \bowtie_F r' = \sigma_F(r \times r')$.

La jointure n'apporte donc rien de nouveau quant aux résultats produits mais elle permet de mettre à des niveaux différents les critère de sélection selon qu'ils sont donnés par la structure ou simplement par la question posée.
La traduction de la jointure en SQL est

```
table1 AS t1 JOIN table2 AS t2 ON t1.attribut1 = t2.attribut2
```

On remarquera qu'on peut renommer les tables jointes afin de simplifier le nom des attributs quand on leur ajoute le nom de la table. L'exemple ci-dessus devient

```
SELECT e.nom, e.classe, n.matiere, n.date, n.note
FROM  e JOIN n ON e.id = n.id_e
```

Dans le cas de plusieurs jointures on peut penser qu'on joint une nouvelle table à une jointure

```
table1 AS t1 JOIN table2 AS t2 ON t1.attribut1 = t2.attribut2
             JOIN table3 AS t3 ON tx.attributa = t3.attribut3
```

ou joindre 3 tables avec 2 conditions associées par AND.

```
table1 AS t1 JOIN table2 AS t2 JOIN table3 AS t3
             ON t1.attribut1 = t2.attribut2 AND tx.attributa = t3.attribut3
```